Journal of Information Processing Vol. 19 180–189 (Apr. 2011)

Regular Paper

Hit-list Worm Detection Using Distributed Sliding Window

Nobutaka Kawaguchi,[†1] Hiroshi Shigeno[†1]
and Ken'ichi Okada[†1]

In this paper, we propose a new distributed hit-list worm detection method: the Anomaly Connection Tree Method with Distributed Sliding Window (ACTM-DSW). ACTM-DSW employs multiple distributed network Intrusion Detection Systems (IDSs), each of which monitors a small portion of an enterprise network. In ACTM-DSW, worm propagation trees are detected by using a sliding time window. More precisely, the distributed IDSs in ACTM-DSW cooperatively detect tree structures composed of the worm's infection connections that have been made within a time window. Through computer-based simulations, we demonstrate that ACTM-DSW outperforms an existing distributed worm detection method, called d-ACTM/VT, for detecting worms whose infection intervals are not constant, but rather have an exponential or uniform distribution. In addition, we implement the distributed IDSs on Xen, a virtual machine environment, and demonstrate the feasibility of the proposed method experimentally.

1. Introduction

In recent years, computer worms have become a serious threat to computer networks. Locating infection targets is an important step for worm propagation. Most existing worms use address scans to locate infection targets, and many techniques for detecting such address scanning have been proposed. Since address scans are obviously anomalous, it is not difficult to detect the activity by monitoring various statistics such as the number of failed connection attempts in an interval of time. However, it is now considered that worms employing more sophisticated target location techniques will emerge in the near future. Among such worms, we have studied worms that acquire the IP address list of vulnerable hosts and attempt to infect all the hosts included in the list [1)]. Such worms are often refereed to as hit-list worms, and are capable of evading most existing worm detection methods based on address scanning. In particular, we focus primarily on the hit-list worms that limit the number of attacks per instance and the propagation speeds in order to evade detection Refs. 1)–2).

Up to now, few works have been done on detecting hit-list worms. Kawaguchi, et al. proposed a method called d-ACTM/VT [2)], for detecting hit-list worms that attack internal hosts in an enterprise network by finding their propagation trees in a distributed manner. A propagation tree represents a worm's propagation routes in a network. More specifically, the tree is a structure composed of infected hosts as nodes and infection connections between the hosts as edges. An infection connection is made when a worm infects another host. Since an infected host usually makes more than one infection connection to other hosts repeatedly, worm's propagation forms a tree structures of infected hosts. d-ACTM/VT concatenates suspicious connections into trees, and judges that the network is being attacked by a worm when the tree size reaches a certain threshold. d-ACTM/VT detects propagation trees in a distributed manner by deploying several fully decentralized distributed Intrusion Detection Systems (IDSs) in a network. Each IDS monitors a part of the network, and propagation trees are detected by exchanging information about suspicious connections with other IDSs. It has been reported that worms can be detected by this method when a few percent of hosts are infected in an enterprise network. Furthermore this approach is scalable.

There is a problem with the approach, however, because suspicious connections are concatenated into trees. Two connections are concatenated if the source or destination host of a connection made earlier is the source of the other, and the difference between their start times is smaller than a certain threshold. This approach is effective against worms with a constant infection interval [⋆1], since all infection connections can be concatenated if the threshold is longer than the constant interval. On the other hand, worms with variable infection intervals conforming to particular distributions (e.g., an exponential distribution) can evade

†1 Faculty of Science and Technology, Keio University

⋆1 There are two types of infection intervals. One is the interval between the time when an infected host makes an infection connection and the time when it makes the next infection connection. The other is the interval between the time when a host is infected by another host and when it makes the first infection connection. In this paper, the term *infection interval* represents both types.

detection by using d-ACTM/VT. This is because even if the average interval is equal to or smaller than the threshold, a certain percentage of actual intervals are longer than the threshold (e.g., if the interval follows a uniform distribution, 50% of the intervals are longer than the average). These relatively long intervals can divide the propagation tree into a few subtrees, which prevents d-ACTM/VT from detecting the worm at an early stage of propagation. Although most existing worms do not change their infection intervals dynamically, researchers have noted the emergence of worms that have variable infection intervals for evading detection [3].

To address this problem with d-ACTM/VT, in this paper we propose a new distributed hit-list worm detection method: the Anomaly Connection Tree Method with Distributed Sliding Window (ACTM-DSW). This approach employs a sliding time window to detect propagation trees. In ACTM-DSW, connections whose start times are within the time window form trees, regardless of the difference between the start times of directly concatenated edges.

More specifically, a new tree is identified each time a new suspicious connection is detected, and then the tree will include some connections that are subsequently made in the next time window. Since ACTM-DSW considers only the difference between the start times of the first and the most recent connections to specify a tree, the intervals between directly concatenated edges are irrelevant. Consequently, this method is less influenced by fluctuations in the infection intervals compared with d-ACTM/VT.

Through computer simulations, we show that ACTM-DSW outperforms d-ACTM/VT in terms of speed for detecting worms whose infection intervals have an exponential or a uniform distribution. In addition, we implement IDSs of ACTM-DSW on the virtual machine environment Xen, and demonstrate the feasibility of the proposed method experimentally.

This paper is organized as follows. In Section 2, d-ACTM/VT, which formes the foundation of ACTM-DSW, is described in more detail. We propose ACTM-DSW in Section 3. In Sections 4 and 5, we report the results of evaluation experiments carried out by computer simulations and our implementation on Xen, respectively. We discuss related works in Section 6. Finally, we present our conclusion in Section 7.

2. d-ACTM/VT

2.1 Anomaly Connections and Anomaly Connection Tree

In this paper, a TCP connection or a bi-directional/unidirectional UDP flow between internal hosts in an enterprise network is referred to as a connection. A connection is identified by a tuple consisting of the source and destination hosts. The source is the host that initiates a connection. Two connections with different contents or port numbers are regarded as equal only if they have both the same source and destination.

d-ACTM/VT detects worm propagation trees by leveraging the difference in the frequency of occurrence between legitimate and infection connections. It is well known that most internal hosts usually communicate with only a small fraction of all internal hosts (e.g., servers) [4)–5)]. Typically, about 80% of connections made by a host are destined to only about less than 20% of all internal hosts. Thus, most legitimate connections made by a host are destined to those with which it has frequently communicated. In contrast, each worm instance tends to selects its infection targets from its hit list, without considering the hosts to which its infected host has made connections. Thus, most infection connections made by an infected host are destined to those with which it has infrequently communicated under normal condition. Throughout this paper, we refer to these infrequently made connections as Anomaly Connections (ACs), and to others as Normal Connections (NCs). For the reasons above, most legitimate connections are classified as NCs. On the other hand, when a worm propagates, most of the infection connections are classified as ACs.

Thus, the idea behind d-ACTM/VT is that most parts of a worm's propagation tree can be mapped into tree structures whose edges are ACs. The trees are referred to as Anomaly Connection Trees (AC trees). A newly made AC is concatenated with an existing one if the gap between their start times is less than a value T_{gap}, and the source of the new connection is either a source or destination of the existing one. **Figure 1** shows an example of an AC tree. If $t_1 - t_0 < T_{gap}$ and $t_2 - t_1 < T_{gap}$ hold, an AC tree formed by hosts A, B, D and E is detected. The tree size is 4. Furthermore, another tree formed by hosts C, F and G is detected if $t_5 - t_4 < T_{gap}$ holds.

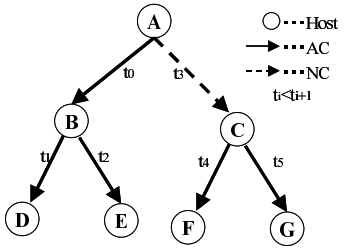

Fig. 1 AC tree.

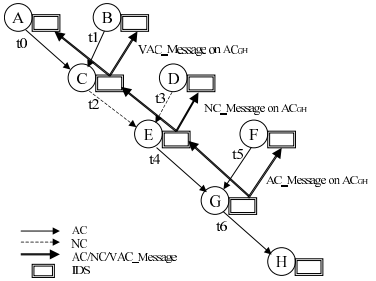

Fig. 2 Transmissions of AC/NC/VAC_Messages.

Each time an AC is observed, an IDS that monitors the source of the AC distributes the information on the AC to other IDSs so that they can detect AC trees in a distributed manner. When the size of the detected AC tree reaches a threshold, an alarm is sent to the network administrator.

2.2 Connection Classification

To classify connections as either ACs or NCs, IDSs maintain a connection history table for each monitored host. This table lists the number of outgoing connections that the monitored host has made to each destination. Then, when a new outgoing connection is detected, it is classified as an NC if the destination is a host with which the source has frequently communicated, corresponding to x% (e.g., 80%) of the total connections in the table; otherwise, the connection is classified as an AC. The table is periodically updated in order to reflect changes in the communication patterns of the monitored hosts over time.

2.3 Virtual Anomaly Connection Tree

Since infection connections are not always classified as ACs (e.g., when an infected host makes infection connections to vulnerable popular servers), worm propagation can generate a few AC trees separated by infection connections classified as NCs. d-ACTM/VT detects the separated propagation trees by clustering these AC trees which are separated by a few NCs. A cluster of AC trees is called as a Virtual AC tree (VAC tree). Then, if the size of a VAC tree, which is the sum of the sizes of clustered trees, reaches a certain threshold, an alarm is raised. In Fig. 1, a tree rooted at host A and a tree rooted at host C are clustered to form a VAC tree if $t_4 - t_3 < T_{gap}$. The VAC tree is 7 in size.

2.4 Communication between IDSs

d-ACTM/VT employs fully decentralized distributed IDSs. Each IDS is responsible for inferring the size of AC/VAC trees that are rooted at its monitored hosts. IDSs exchange information on monitored connections so that each can accurately infer the size of the trees. While there are several types of messages used in d-ACTM/VT, here we briefly describe one type used for inferring AC tree size. More details on the messages are given in Ref. 2).

When an IDS detects that a monitored host X has made an AC, it sends the information via messages to other IDSs monitoring hosts that have made ACs to X within T_{gap}. IDSs that receive the message further propagate it to other IDSs in a similar manner. Let IDS_X denote an IDS that monitors a host X, and let AC_{XY} denote an AC made from hosts X to Y. Now assume that AC_{AB} and AC_{BD} are made in Fig. 1, and $0 \leq t_1 - t_0 \leq Tgap$ holds. As a result, a tree composed of hosts A, B and D is detected. Note that since IDS_A does not observe AC_{BD}, it cannot directly infer the actual tree size (= 3). Then, IDS_B sends IDS_A a message indicating that AC_{BD} is made at t_1. This message enables IDS_A to infer the actual size of the tree. Likewise, if host D later makes an AC within T_{gap} after t_1, IDS_D sends a message to IDS_B. In this case, IDS_B relays the message to IDS_A, and IDS_A updates the inferred size from 3 to 4. Since each IDS does not require a central server for obtaining global knowledge of the network and analyzing connection logs of all internal hosts, this approach is scalable [2)].

2.5 Problem with d-ACTM/VT

As stated in Section 1, there is a problem with the manner that d-ACTM/VT connects ACs into AC trees. Although the approach is effective at finding propagation trees of worms that make new infection connections periodically every T_{gap} or less, if a worm's infection interval is not a constant but rather a variable conforming to a certain probability distribution with large variance (e.g., an exponential distribution), a considerable number of infection connections cannot be joined even if the average interval is less than T_{gap}. Therefore, it is difficult for d-ACTM/VT to detect all parts of the propagation tree for such worms.

A straightforward solution is to set T_{gap} to a larger value. For example, if

T_{gap} is set to several times of the average infection interval, most of the actual infection intervals will be less than Tgap. As T_{gap} is increased, however, AC trees tend to become larger, and the tree size threshold must be increased to prevent an increase in the false positive rate. As a result, this may cause considerable degradation in the detection speed.

3. ACTM-DSW

3.1 Concept

In this section, we propose a new distributed worm detection method, called ACTM-DSW. ACTM-DSW employs fully-decentralized distributed IDSs so that this is as scalable as d-ACTM/VT, and it uses a sliding time window of size T_w for tree detection. In ACTM-DSW, connections that have been made within the past T_w form AC and VAC trees at the current time. Two ACs within the window are connected if a source or destination of the preceding AC is a source of the other. Accordingly, a new AC can be added to an AC tree if the AC and one of the tree's edge satisfy the condition. Therefore, this approach can cope with worms with variable infection intervals. In Fig. 1, if $t_2 - t_0 < T_w$ holds, hosts *A, B, D* and *E* form an AC tree. Furthermore, if $t_5 - t_4 < T_w$ holds, hosts *C, F* and *G* form an AC tree. Likewise, if $t_5 - t_0 < T_w$ holds, an AC tree of {*A, B, D, E*} and an AC tree of {*C, F, G*} form a VAC tree.

In d-ACTM-DSW, when a new AC is detected, a new tree with the AC as the first edge is recognized. Then, the tree grows and is later removed as time advances. Here, $AT(E)$ denotes an AC tree whose first AC is E, and then $VAT(E)$ denotes a VAC tree whose first edge is E. In Fig. 1, AC_{BD} is made at t_1. The AC forms a new tree $AT(AC_{BD})$ and is also a part of $AT(AC_{AB})$. Then, at $t_0 + T_w$, $AT(AC_{AB})$ is removed. At t_2, AC_{BE} is added to $AT(AC_{BD})$ if $t_0 + T_w < t_2 < t_1 + T_w$ holds. Finally, $AT(AC_{BD})$ is removed at $t_1 + T_w$. The maximum size of $AT(AC_{BD})$ is 3. ACTM-DSW raises an alert when the size of an AC tree reaches a certain threshold. These dynamics of tree topology hold true for VAC trees.

The challenging issue with ACTM-DSW is how the distributed IDSs infer the size of trees composed of connections started within the past T_w. ACTM-DSW addresses this issue by propagating the information on a newly detected AC to

```
procedure respondtoAC(new_ac)
   //add a new AC to existing trees
   for out_ac in outbound_ac_list
      if(AT(out_ac).add(new_ac)==SUCCEED)
         if(AT(out_ac).size >= TH_AC)
            raiseAlert()
   //add a new AC to an outbound AC list
   outbound_ac_list.add(new_ac)
   //send AC_Message
   for in_ac in inbound_ac_list
      if(ac.time-Tw<=in_ac.time)
         ids=locateIDS(in_ac.src)
         send_ACMessage(ids,new_ac)

procedure receiveAC_Message(new_ac)
   //add a new AC to existing trees
   for out_ac in outbound_ac_list
      if(AT(out_ac).add(new_ac)==SUCCEED)
         if(AT(out_ac).size >= TH_AC)
            raiseAlert()
   //send AC_Messages
   for in_ac in inbound_ac_list
      if(new_ac.time-Tw <= in_ac.time <= out_ac.time)
         ids=locateIDS(in_ac.src)
         sendAC_Message(ids, new_ac)
```

Fig. 3 Algorithm to transmit AC_Messages.

IDSs monitoring hosts that have made connections within the past T_w. Then, on the basis of this information, each IDS infers the size of the trees whose first edges are formed by its monitored hosts. In the following sections, we explain how the IDSs detect AC/VAC trees. For simplicity, we assume that each IDS monitors only a single host.

3.2 Distributed AC Tree Detection

ACTM-DSW uses three types of messages for transmitting information on ACs: $AC\_Messages$, $NC\_Messages$ and $VAC\_Messages$. Here, we show how IDSs infer AC tree size by exchanging $AC\_Messages$. **Figure 2** shows an example of message transmission triggered by AC_{GH} made at t_6. For simplicity, we assume that $t_6 - t_0 < T_w$ holds. In response to the occurrence of AC_{GH}, IDS_G sends $AC\_Messages$ on AC_{GH} to the IDSs that monitor hosts E and F, which made ACs with host G within T_w. **Figure 3** shows in detail how the IDSs act.

Upon observing that a monitored host makes an outbound AC, the IDS calls **respondtoAC** with the newly detected AC as the argument. This function adds the new AC to $AT(X)$ which appeared within the past T_w, and raises an alert if the tree size reaches TH_{AC}. Here, X is included in **outbound_ac_list**, a list of all outbound ACs the monitored host has ever made. The new AC is added to the list. Then, the IDS transmits $AC\_Messages$. The receiver is an IDS monitoring a host that has made a connection to the sender's monitored host within T_w. Here, how to locate the addresses of the receiver IDSs (IDS_E and IDS_F in Fig. 2) is an issue. While various approaches can be considered, we introduce a

```
procedure respondtoAC+ (new_ac)
  for out_ac in outbound_ac_set
    //add a new AC to existing VAC trees
    if(VAT(out_ac).add(new_ac)==SUCCEED)
      if(VAT(out_ac).size >= TH_VAC)
        raiseAlert()
    //send NC_Messages
    if(AT(out_ac).contains(new_ac))
      if(AT(out_ac).size > TH_PROPAGATE)
(1)     for in_nc in inbound_nc_list
          if(new_ac.time-Tw<=in_nc.time<=out_ac.time)
            ids=locateIDS(in_nc.src)
            sendNC_Message(new_ac,hop=1, in_nc.time)
      if(AT(out_ac).size == TH_PROPAGATE)
        for ac in AT(out_ac)
(2)       for in_nc in inbound_nc_list
            if(ac.time-Tw<=in_nc.time<=out_ac.time)
              ids=locateIDS(in_nc.src)
              sendNC_Message(ac,hop=1, in_nc.time)

procedure receiveNC_Message(ac, hop, max_time)
  //send NC_Messages
  if(hop < MAX_HOP)
    for in_nc in inbound_nc_list
      if(ac.time-Tw<=in_nc.time<=max_time)
        ids=locateIDS(in_nc.src)
        sendNC_Message(ac,hop+1,in_nc.time)
  processVAC (ac)

procedure processVAC(ac)
  //add a AC to existing VAC trees
  for out_ac in outbound_ac_list
    if(VAT(out_ac).add(ac)==SUCCEED)
      if(VAT(out_ac).size >= TH_VAC)
        raiseAlert()
      //send VAC_Messages
      for in_ac in inbound_ac_list
        if(ac.time-Tw <= in_ac.time <= out_ac.time)
          ids=locateIDS(in_ac.src)
          sendVAC_Message(ids, ac)
```

Fig. 4 Algorithm to transmit NC/VAC_Messages.

solution in Section 5.

Upon receiving an $AC\_Message$, the IDS calls **receiveAC_Message** and updates tree sizes accordingly. In Fig. 2, IDS_E receives an $AC\_Message$ from IDS_G, and $AT(AC_{EG})$ is updated to 3. Then, the IDS further propagates the $AC\_Message$ to other IDSs recursively if its monitored host has received ACs within T_w. If the average interval of legitimate connections is a for every host and the ratio of ACs to all connections is b, $E(AC\_Message)$, the expected number of transmitted $AC\_Messages$ each time a new AC is detected, can be expressed as follows:

$$E(AC\_Message) = e^{\frac{b \times T_w}{a}} - 1. \quad (1)$$

This implies that $E(AC\_Message)$ is exponentially proportional to T_w.

3.3 Distributed VAC Tree Detection

NC/VAC messages are used for transmitting information on an AC tree to other AC trees that are separated by a few NCs in order for these trees to be clustered into a VAC tree. In Fig. 2, $AT(AC_{AC})$ and $AT(AC_{EG})$ (including hosts E, G and H) are separated by NC_{CE}, and can be clustered into $VAT(AC_{AC})$. **Figure 4** presents the details. In VAC tree detection, when an IDS detects that its monitored host has made an outbound AC, **respondtoAC+** is called after **respondtoAC/receiveAC_Message** is completed. In the function, the AC is concatenated with existing VAC trees; if the size of a tree reaches TH_{VAC}, an alert is raised.

Next, if the new detected AC is concatenated with an AC tree that is larger than a threshold called $TH_{PROPAGATE}$, the IDS calls **sendNC_Message**, and then sends $NC\_Messages$ on the AC to IDSs whose monitored hosts have made an NC to the sender's monitored host NCs (**respondtoAC+(1)**). In Fig. 2, if $TH_{PROPAGATE} = 2$, since $AT(AC_{EG}) = 3$, $NC\_Messages$ on AC_{GH} are sent from IDS_E to IDS_C and IDS_D.

On the other hand, if the size of an AC tree reaches $TH_{PROPAGATE}$ upon adding the AC, an $NC\_Message$ on every AC in the tree is transmitted so that one AC tree is concatenated to other AC trees beyond the intermediate NCs (**respondtoAC+(2)**). In Fig. 2, if $TH_{PROPAGETE} = 3$, an $NC\_Message$ on AC_{EG}, as well as a message on AC_{GH}, is transmitted to IDS_C and IDS_D, which then recognize $AT(AC_{EG})$ for the first time.

Upon receiving an $NC\_Message$, the receiver calls **receiveNC_Message**, and then the message can be further propagated recursively. Note that in contrast to $AC\_Messages$, the number of maximum hops of an $NC\_Message$ is limited to a value MAX_HOP. When MAX_HOP is set to 1, each $NC\_Message$ can takes 1 hop, and therefore AC trees separated by one NC can be clustered to a VAC tree.

After transmitting $NC\_Messages$, the IDS calls **processVAC**. In **processVAC**, an AC on the received $NC\_Message$ is added to existing VAC trees, and the tree sizes are evaluated. Then, $VAC\_Messages$ on the AC are propagated, as in the case of $AC\_Message$ transmission. In contrast with $AC\_Messages$, ACs on $VAC\_Messages$ are added to VAC trees, which results in the clustering of AC trees. In Fig. 2, $VAC\_Messages$ on AC_{GH} are sent to IDS_A and IDS_B.

Upon receiving a $VAC\_Message$, the receiver calls **processVAC**. Then, VAC trees sizes are updated and evaluated. In Fig. 2, $VAT(AC_{AC})$, which is composed of $AT(AC_{AC})$ and $AT(AC_{EG})$, is evaluated at IDS_A. Finally the message is further propagated recursively.

4. Evaluation Experiment

We evaluate the detection performance of ACTM-DSW in computer simulations and compare the proposed method with d-ACTM/VT.

4.1 Simulation Conditions

We simulate an enterprise network where all internal hosts are vulnerable. Then, a hit-list worm intrudes into the network and attempts to infect all the internal hosts by exploiting the vulnerability. Each distributed IDS monitors one host. Thus, the number of IDSs is equal to the number of hosts. Details on the simulation condition are as follows.

4.1.1 Network Model

We simulate a typical enterprise network where common server-client type network services such as SSH, Windows RPC, Web and E-mail are operating.

Since the performance of ACTM-DSW is affected by a pattern of the destinations and the intervals of the legitimate connections, we use the pattern as a network model for evaluation. The simulated network has 500 internal hosts, a typical size for an enterprise network [5]. Among the hosts, 20% are server-hosts which are frequently accessed by other hosts, and the remaining 80% are client-hosts which are infrequently accessed by other hosts. In our communication model, 80% of the legitimate connections that each host makes are destined to server-hosts, and the remaining 20% are destined to client-hosts. References 4)–5) report that most hosts in an enterprise network communicate frequently with a small number of other hosts. Thus, destinations of legitimate connections in our model are less biased than those of typical enterprise networks. Note that, Ref. 2) reports that as less biased the destinations of legitimate connections are, the more hosts that worms can infect before detection. Thus, for the purpose of detection, the simulation condition is harder than typical enterprise networks.

Each host makes legitimate connections to other hosts periodically, and the interval time of legitimate connections made is an exponential variable with a mean of 10 TU (TU is a time unit). To evaluate the characteristic of ACTM/DSW under a basic and primitive network model, we assume that all hosts have the same the average interval. We will treat a case where each host has a different average interval in future works.

The lengths of other time-related parameters are represented from the viewpoint of how much larger they are compared to the average interval of legitimate connections. Since the average interval that a host in enterprise networks makes TCP connections is tens of seconds [5], 1 TU corresponds to a few seconds.

4.1.2 Worm Model

Our primary concern is a hit-list worm that limits the number of infection trials and infection speeds in order to evade connection-rate based detections [6], as stated in Ref. 1).

The worm has a hit-list that includes all addresses of the internal hosts. Each worm instance selects target hosts from the hit-list in order. The hit-list worms share a hit-list in a way such that no hosts are attacked more than once [13)–14)]. The hit-list worm is modeled with the number of infection attempts and the infection intervals. The number of infection attempts of each worm instance is limited to 2, which is almost minimum and low enough for worms to evade connection-rate detections Ref. 1).

The average infection interval is set to 10 TU, equal to the average interval of legitimate connections. As most existing worms make hundreds of connections in a second [14], the average interval is quite relatively long. In the simulation, the infection interval is set to a constant, an exponential or a uniform variable. With a constant distribution, the infection interval is always 10 TU. With uniform distribution, the infection interval is in the range 0-20 TU.

In summary, when a host is infected, it starts to attack other hosts tens of seconds (= 10 TU) after the infection on average. Then, it makes infection connections twice, and the interval of the connections is ten of seconds on average. Thus, the propagation speed of the worm is much slower than that of a Flash worm [1]. We also evaluate the performance of ACTM-DSW against worms that spread fast like a Flash worm.

4.1.3 IDS Settings

In ACTM-DSW, IDSs classify 80% of the top legitimate connections as NCs. The ratio is optimized to classify the types of connections specified by the model in Section 4.1.2. Deriving the optimized ratio based on network conditions is one of our future works.

The target value of the false alert interval is 10,000 TU on average. With this

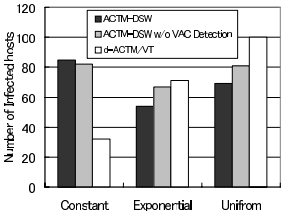

Fig. 5 Number of infected hosts for three types of intervals.

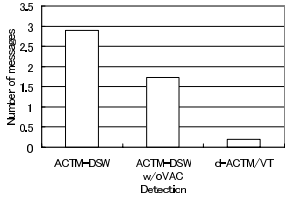

Fig. 6 Number of transmitted messages per AC.

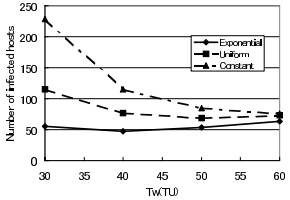

Fig. 7 Effect of T_w on number of infected hosts.

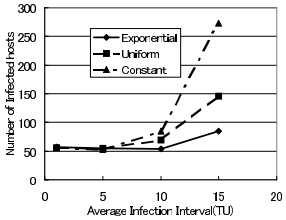

Fig. 8 Effects of infection interval on number of infected hosts.

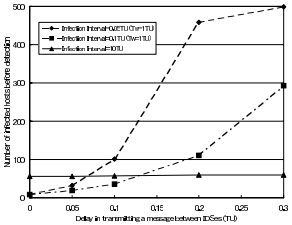

Fig. 9 Effect of communication delays on the number of infected hosts.

interval, there are a few false alerts a day, and the frequency is considered to be small enough [9]. TH_{AC}/TH_{VAC} are adjusted to values that satisfy the interval.

T_w is set to 50 TU, which is considered to be a reasonable value. In the evaluation, we assess the effect of T_w on the detection performance. TH_{AC} is adjusted to 30 to achieve the given false alert interval using the threshold-adjusting algorithm [2]. $TH_{PROPAGATE}$ is set to 10, since it is reported that 35% of TH_{AC} is best for $TH_{PROPAGATE}$ in d-ACTM/VT [2]. Likewise, MAX_HOP is set to 1, which will be best in d-ACTM/VT [2]. In a similar manner to TH_{AC}, TH_{VAC} is adjusted to 40. In this experiment, T_{gap}, which is a parameter of d-ACTM/VT, is set to the optimal value such that the infected hosts are minimized.

Finally, by default, we consider the delay in transmitting a message between IDSs is ignorable, and set the delay to 0 TU. This is because the delays are quite small compared to the infection interval of the simulated worms. When evaluating our results, we discuss the case where the delay cannot be ignored for detecting fast-spreading worms.

4.2 Results

Figure 5 shows the number of infected hosts before an alert is raised for d-ACTM/VT, ACTM-DSW and ACTM-DSW without VAC tree detection (only AC tree detection is considered, and the parameters are set to raise an alert every 10,000 TU). Against a worm with a constant infection intervals distribution, the number of infected hosts for ACTM-DSW is more than twice that for d-ACTM/VT. On the other hand, against worms with infection intervals with an exponential or uniform distribution, the number of infected hosts for ACTM-DSW is up to 30% less than that for d-ACTM/VT. Furthermore, without VAC tree detection, the number of infected hosts increased by up to 20%, which demonstrates the effectiveness of VAC tree detection. For each detection method, the number of infected hosts is less for infection intervals having an exponential distribution compared with a uniform distribution. In the case of an exponential distribution, about 60% of intervals are shorter than the average, while 50% of intervals are shorter in the case of a uniform distribution. This difference in the percentage of intervals that are shorter than the average helps explain the observed results.

Figure 6 shows the average number of messages transmitted in the response to a newly detected AC. ACTM-DSW requires more than 10 times as many messages as d-ACTM/VT. The average number of AC_Messages is 1.7, which fits Expression (1).

Figures 7 and **8** show the effects of T_w and the worm's average infection interval on detection performance, respectively. It can be seen in Fig. 7 that too small a value of T_w results in a higher number of infected hosts. In Fig. 8, as the average infection interval increases, more hosts are infected.

Next, we evaluate ACTM-DSW against fast worms. As stated in Section 4.1.3, against fast worms, the delay in transmitting a message between IDSes cannot be ignored. **Figure 9** shows the number of infected hosts as a function of the communication delay between IDSes. We simulate three types of worms with infection intervals of 0.05, 0.1 and 10 TU. The number of infection trials of each copy is 2 for all worm types in order to evade connection-rate based detections [6].

We call the worms with infection intervals of 0.05/0.1 TU as fast hit-list worms. We designed hosts infected by these worms to make two infection connections at almost the same time 0.05/0.1 TU after the infection, since most existing worms make multiple connections concurrently. In Ref. 14), the interval between the time when the Flash worm attacks a host and the time when the attacked host begins infection activities is modeled as 2 seconds. Since 0.05/0.1 TU are shorter than, or comparable to 2 seconds, the intervals of the fast-hit-list worms fit the Flash worm model [14]. To detect the worms that construct large AC trees in a short time, we set T_w to 1 TU. Accordingly TH_{AC} and TH_{VAC} are adjusted to about 4 and 5 respectively.

As for the worm with an infection interval of 10 TU, the number of infected hosts is almost stable as the delay gets longer, which indicates that the effect of the transmission delay is quite small. As for the fast-hit-list worm with an infection interval of 0.05 TU, on the other hand, the number of infected hosts is significantly increased, as the delay gets longer. When the delay is 0.05 TU, the number of infected hosts is 32. The number is increased to 100 when the delay is 0.1 TU. Likewise, for worms with infection intervals of 0.1 TU, the number of infected hosts increases steeply as the delay exceeds 0.1 TU.

The results demonstrate that ACTM-DSW detects such fast-hit-list worms faster than it detect worms with infection intervals of 10 TU, when the delay in transmitting a message between IDSes is equal to, or shorter than the interval between when a host is being attacked and when the host starts to infect others.

4.3 Discussion

The results show that ACTM-DSW and d-ACTM/VT compensate for each other: against worms with constant infection intervals, d-ACTM/VT exhibits higher performance, while against worms with infection intervals of high variance, ACTM-DSW does. Thus, the combination of the two methods can be effective against a wide range of worms.

A drawback of ACTM-DSW is that a large number of messages must be transmitted for detection. In future work, we will attempt to address this problem by employing a sampling algorithm to reduce the number of transmitted message without degrading detection performance.

5. Implementation of Distributed IDS on Xen

In this section, we describe the implementation of distributed IDSs on Xen and evaluate the performance. First, we introduce how an IDS locates the addresses of other IDSs. Then, the implementation details and evaluation results are presented.

5.1 Locating the IP Addresses of Distributed IDSs

To send messages to an IDS that monitors an arbitrary host, the sender must locate the address of the IDS. In addition, confidentiality and integrity must be satisfied.

A simple method for locating IDSs is to deploy a server that maps the monitored hosts and IDSs. The approach, however, is not scalable and makes the server a single point of failure. Therefore, this is not a suitable approach for distributed detection systems.

Instead, as shown in **Fig. 10**, we introduce a 3-way protocol for locating IDSs and sending messages, taking advantages of the fact that each IDS captures all the traffic to and from its monitored hosts. This protocol assumes that each IDS has a private/public key pair and a certificate signed by a closed PKI that authenticates IDSs implementing ACTM-DSW in the network.

Figure 10 shows message transmission from IDS_A to IDS_B. First, IDS_A sends a packet that includes a message ID and its certificate to *hostB* via a UDP packet. Since IDS_B monitors incoming traffic to host B, it can capture the packet. IDS_B checks that the message is sent to its monitored host and conforms to the protocol message format. Then it sends back a packet that includes the same message ID and its certificate to IDS_A. Upon receiving the

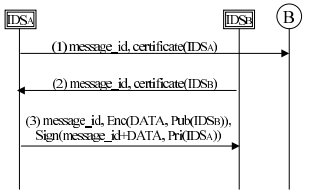

Fig. 10 Message transmission protocol.

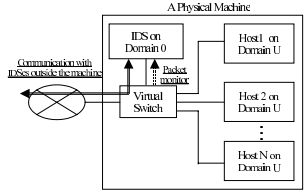

Fig. 11 Implementation of distributed IDS on Xen.

response from IDS_B, IDS_A receives the public key of IDS_B. Then, IDS_A sends IDS_B a packet that includes the same message id, a message encrypted by IDS_B's public key, a signature of the message ID and the message encrypted by IDS_A's private key. Finally, IDS_B verifies and processes the message.

5.2 Implementation on Xen

We implement a distributed IDS on the virtual environment Xen. **Figure 11** shows the implementation of the IDS on a Xen virtual machine. This IDS is written in JDK 1.6 and has about 3,000 lines of source code. For encryption and signing, 1024-bit RSA with SHA-1 is used.

This IDS runs on domain-0, which is a privileged domain, and monitors the network activity of virtual machine hosts running on Domain-U in the same physical machine. Since Domain-0 has full control over virtual switches, the IDS can capture all traffic to and from the virtual hosts through the switches.

5.3 Evaluation Experiment

5.3.1 Environment

To evaluate the performance of the IDSs on Xen, we conduct the following experiment. We use a small network composed of two Xen machines. Each machine has an Intel Core 2 Quad processor at 2.66 GHz, 3 GB of memory, CentOS 5.2 and Xen 3.2. Each machine contains an IDS and five virtual hosts, which can communicate with hosts in both machines. Of the virtual hosts in each machine, one is a server-host and the others are client-hosts. Each host routinely sends 1 MB of data at an average interval of 10 seconds, and 80% of the destinations are server hosts. IDS parameters are set to values such that one false alert is raised on average every 20 minutes.

For the worm propagation scenario, we created a program that mimics a hit-list worm. This program makes an infection connection to a host listed in the hit list every 1 s. We run this dummy worm to measure the number of infected hosts when an IDS raises an alert.

5.3.2 Resuts

As for the detection speed, about 6-7 hosts are infected when an IDS raises an alert. Although more than half of all hosts are infected before detection, we are not discouraged by this result since ACTM-DSW is intended to be used on a network with several hundreds of hosts. The computer simulation shows a similar result. As for the computation cost, the CPU usage rate of each IDS is no more than 3%. As for the communication cost, about 2 KB of data are required to transmit each message between the IDSs. Thus, the IDS communication traffic is less than 100 Kbps. These results demonstrate that this IDS is lightweight, and can be deployed in various environments.

6. Related Work

Several researchers have proposed methods that detect worms by monitoring propagation trees or chains composed of infection connections. Staniford-Chen, et al. first introduced the notion of a network activity graph (tree) for detecting worm propagation in organization networks. Their method, called GrIDS [7], detects worms when a graph that meets a predefined rule is constructed. GrIDS employs hierarchical IDSs. For detecting graphs spreading over wide areas, information monitored by lower-layer IDSs is passed to higher-layer IDSs. Ellis, et al. proposed a connection tree detection method that uses a time window [8)–9)]. Although these methods perform connection tree detection, neither takes advantage of anomaly connections for graph (tree) detection. Toth, et al. proposed a method that searches for a chain of anomaly connections and raises an alert if the chain includes connections that contain similar payloads [10)–11)]. Patrick, et al. also proposed a hit-list worm detection method in which connection graphs are constructed from both the attacking hosts outside of the network and attacked hosts within the network [12)].

In contrast to these approaches above, ACTM-DSW detects two types of trees composed of anomaly and normal connections through the cooperation of fully decentralized distributed IDSs. It also need not inspect packet payloads.

7. Conclusion

In this paper, we have proposed ACTM-DSW, a new distributed worm detection method that is effective against hit-list worms with variable infection intervals. ACTM-DSW employs fully decentralized IDSs that cooperatively detect worm propagation trees in a distributed manner. Through computer simulations, we have shown that ACTM-DSW outperforms d-ACTM/VT in detecting worms with infection intervals having exponential and uniform distributions. We have

also implemented the decentralized distributed IDS on Xen, and demonstrated the feasibility of the proposed detection method.

As stated in the discussion, ACTM-DSW requires a large number of message transmissions between IDSs, which is exponentially proportional to the time window size. Decreasing the number is planned for the future work.

References

1) Staniford, S., et al.: The Top Speed of Flash Worms, *Proc. ACM WORM 2004*, pp.33–42 (2004).
2) Kawaguchi, N., et al.: d-ACTM/VT: A Distributed Virtual AC Tree Detection Method, *IPSJ Journal*, Vol.49, No.2, pp.1010–1021 (2008).
3) Yu, W., et al.: On detecting camouflaging worm, *Proc. ACSAC 2006* (2006).
4) Aiello, W., et al.: Analysis of Communities of Interest in Data Networks, *Proc. Passive and Active Measurement Workshop 2005* (2005).
5) Pang, R., et al.: A First Look at Modern Enterprise Traffic, *Proc. the Internet Measurement Conference* (2005).
6) Williamson, M.: Throttling Viruses: Restricting Propagation to Defeat Malicious Mobile Code, Technical Report HPL-2002-172 (2002).
7) Staniford-Chen, S., et al.: GRIDS: A Graph-Based Intrusion Detection System for Large Networks, *Proc. 19th National Information Systems Security Conference*, pp.361–370 (1996).
8) Ellis, D., et al.: A behavioral approach to worm detection, *Proc. ACM WORM 2004*, pp.43–53 (2004).
9) Ellis, D., et al.: Graph based worm detection on operational enterprise networks, MITRE Technical Report (2006).
10) Toth, T., et al.: Connection-history based anomaly detection, *Proc. 3rd IEEE Information Assurance Workshop* (2002).
11) Al-Hammadi, Y., et al.: Anomaly detection for Internet worms, *Proc. 9th IEEE/IFIP IEEE International Symposium on Integrated Network Management*, pp.133–146 (2006).
12) Patrick, M., et al.: Hit-List Worms Detection and Bot Identification in Large Networks Using Protocol Graphs, *Proc. RAID* (2007).
13) Staniford, S., et al.: How to own the Internet in your spare time, *Proc. 11th USENIX Security Symposium* (2002).
14) Zou, C.C., et al.: On the performance of Internet worm scanning strategies, *Performance Evaluation*, Vol.63, pp.700–723, Elsevier (2006).

(Received February 15, 2010)
(Accepted January 14, 2011)
(Released April 6, 2011)

Nobutaka Kawaguchi received his B.S. degree from the Department of Information and Computer Science at Keio University, Japan in 2003, and M.S. and Ph.D. degrees in Open and Environment Systems from Keio University in 2005 and 2008. His current research interests include network security and malware analysis. He received the FUNAI Best Paper Award in 2008. He is a member of IEEE and ACM.

Hiroshi Shigeno received his B.S., M.E. and Ph.D. degrees in instrumentation engineering from Keio University, Japan in 1990, 1992 and 1997. Since 1997, he has been with the Department of Information and Computer Science at Keio University, where he is currently an associate professor. His current research interests include computer networking architecture and protocols, mobile and ubiquitous computing, and Intelligent Transport Systems. He is a member of IEEE and ACM.

Ken'ichi Okada received his B.S., M.S. and Ph.D. degrees in instrumentation engineering from Keio University, in 1973, 1975, and 1982, respectively. He is currently a professor in the Department of Information and Computer Science at Keio University. His research interests include CSCW, groupware, human computer interaction, and ubiquitous computing. He is a member of IEEE, ACM, IEICE. Dr. Okada received the IPSJ Best Paper Award in 1995, 2000, and 2008, the IPSJ 40th Anniversary Paper Award in 2000, IPSJ Fellow in 2002 and the IEEE SAINT Best Paper Award in 2004.